Enriching the Gene Ontology via the Dissection of Labels using the Ontology Pre-Processor Language

Jesualdo Tomas Fernandez-Breis, Luigi Iannone, Ignazio Palmisano, Alan L. Rector, and Robert Stevens

Departamento de Informatica y Sistemas
Facultad de Informatica
Universidad de Murcia
CP 30100, Murcia, Spain
jfernand@um.es
School of Computer Science
University of Manchester
Oxford Road
Manchester
United Kingdom
M13 9PL
lastname@cs.manchester.ac.uk

Abstract. In this paper we describe the process of taking an axiomatically lean ontology and enriching it through the automatic application of axioms using ontology design patterns (ODP). Our exemplar is the Gene Ontology's Molecular Function Ontology; this describes an important part of biology and is widely used to describe data. Yet much of the knowledge within the GO's MF is captured within the term's that label the concepts and within the natural language definitions for those concepts. Whilst both of these are absolutely necessary for an ontology, it is also useful to have the knowledge within the textual part of the ontology exposed for computational use. In this work we use an extension to the Ontology PreProcessor Language (OPPL) to dissect terms within the ontology and add axiomatisation, through OPPL's application of ODP, that make the knowledge explicit for computational use. We show the axiomatic enriching of the GO MF; that this can be accomplished both rapidly and consistently; that there is an audit trail for the transformation; and that the queries supported by the ontology are greatly increased in number and complexity.

1 Introduction

In this paper we describe the automatic transformation of an axiomaticaly lean ontology to one that is axiomatically richer through the application of ontology design patterns (ODP). We do this by exploiting the semantics within the labels on classes and the relatively systematic and explicit nature of naming within

the ontology in question [1] and related ontologies. As the authoring of large, axiomatically rich ontologies by hand, whether *de novo* or by enhancing an existing ontology, involves considerable effort, the ability to automatically enrich ontologies by 'dissecting' their labels and generating the axioms they imply is attractive. Also, ontologies, like software, will need re-factoring during development as testing reveals problems and conceptualisations of the domain change. Applying changes consistently across an ontology by hand is difficult, prone to errors of commission and omission—particularly inconsistency between two changes of the same 'type'.

Many ontologies have much information within the labels on concepts that are not apparent axiomatically for computational inference. One such ontology is the molecular function (MF) aspect of the Gene Ontology (GO) [2]. GO MF describes the activities that occur at the molecular level for gene products such as proteins. There is a great deal of knowledge about the entities described within the labels and text definitions held on classes; these are useful for human users of GO, but not much good for machine processing. For example, as biochemists we can see that the label "vitamin binding" means the binding of a vitamin (and that a vitamin is a chemical) and that 'isoprenoid binding' means the binding of the chemical isoprenoid. However, the ontology does not contain an axiomatic description of what either binding or vitamin means. Similarly, 'oxidoreductase activity, acting on peroxide as acceptor' captures a considerable amount of knowledge in the function's label, but not axiomatically. Consequently, the reasoner cannot take advantage of such knowledge.

If we analyse the structure of the labels of the families of functions in GO, we could identify some regularities. Most binding functions have a label with the structure 'X binding', where X is a biochemical substance. Most types of structural molecule activity have a label with the structure "structural constituent of Y", where Y is a macromolecular complex. Given such regularity, we should be able to systematically pull out patterns of axioms that can make the semantics explicit. We explore a technique for using these naming conventions within labels as a way of mapping to patterns of axioms that make the information in the label computationally explicit.

Ontology Design Patterns (ODPs) are templates that define best practices in Ontology Engineering [3]. An ODP can then be viewed as a template that can be systematically applied for the representation of a given situation.

The Ontology Preprocessor Language (OPPL) is a scripting language for OWL that can be used to apply ODP across an ontology [4,5]; it is a way of programmatically manipulating an ontology at a higher level of abstraction than, for instance, the Java OWL API (http://owlapi.sourceforge.net/). It has the typical advantages of consistent, rapid application of the patterns. If the patterns to be applied change, those changes can also be rapidly applied. Thus, the OPPL scripts act as a documentation of the changes applied to the ontology.

We have extended OPPL2 to include regular expressions that will capture patterns within labels on concepts. This means we can *dissect* a label and use the elements within the label to populate a pattern that has been conceptualised

to match a particular pattern of elements within a label. This technique has been used before [6,7] by using bespoke programmes to do the dissection and application of axioms. Here we present a generic tool for this technique and describe its application to a large ontology.

2 Materials and Methods

The method applied in the transformation is:

1. Inspect current GO molecular function labels and text definitions to find out what needs to be revealed as axioms;
2. Develop patterns of axioms that capture the knowledge about the concepts, relating the need for axioms to elements within the term label for the concept.
3. Identify supporting ontologies or modules that capture entities within the developed patterns;
4. Apply patterns across source ontology using OPPL2 to both apply ODP and to recognise lexical patterns in labels;
5. Run a reasoner and inspect the resulting ontology.

Analysing the GO Molecular Function Ontology: In this work we have used the version 1.550 of this ontology, which was downloaded on 19th April 2009 from http://www.geneontology.org This ontology had 8 548 classes, 5 object properties, 5 data properties and 9954 subclass axioms. We transformed only some parts of the GO MF ontology: binding; structural molecule activities; chaperone activities; proteasome regulator activities; electron carrier activities; enzyme regulator activities and translation regulator activities. The labels of the classes within each of these areas were inspected and the patterns from the naming conventions extracted. In addition, the text definitions of the class were inspected to help elucidate the patterns.

Producing the ODP: The analysis of the names and text definitions identifies what needs to be made explicit; this stage prescribes how those semantics will be made explicit as a set of axioms. The underlying technique is that of *normalisation* [8]:

- A tree is formed along the primary axis of classification; in this case the functions.
- Other axis of classification, such as chemical type, are separated into *supporting ontologies*.
- A restriction can be formed from the tree of functions of the supporting ontology to capture the aspects separated out into the supporting ontologies. For example, *vitamin binding* might have the restriction *binds some vitamin*.

As described in Section 1, the knowledge held within the labels, the text definitions of each class and the background knowledge of the ODP creator are combined to develop the ODP. The relationships used were taken from OBO's relationship ontology (RO) [9] to retain consistency with OBO.

The auxiliary ontologies: The creation of the ODP suggests the need for a range of supporting ontologies. The application of the ODP strongly depends on the availability of these ontologies. Wherever possible, we used ontologies already developed—especially those from the OBO Foundry http://www.obofoundry.org/, the original creators of the GO. Otherwise, or if the chosen ontologies were incomplete, sufficient ontology was created to fulfil the needs of this work.

Application of ODP using OPPL: The Ontology Pre-Processing Language's main motivation was to provide a declarative language for: (a) Specifying how an ontology should be modified; (b) Optionally specifying the conditions under which such modifications should be enacted [4,5]. A generic OPPL 2 statement looks like:

```
 <PREAMBLE> SELECT Axiom,...,Axiom BEGIN ADD
| REMOVE Axiom ... ADD | REMOVE Axiom END
```

An OPPL 2 statement is decomposable into the following sections:

1. Variables (before `SELECT`): here all the variables to be used in the following sections must be declared.
2. Selection (between `SELECT` and `BEGIN`): here a set of axioms are specified using the variables declared in the previous section. The resulting matches will consist of all those axioms[1] that hold in the knowledge base under consideration, for every possible variable substitution.
3. Actions (between `BEGIN` and `END`): here a set of axioms to be added or removed are specified. Such axioms could also depend on variables, but only on those bound by one or more select clauses.

The OPPL grammar is available at http://oppl2.sourceforge.net/grammar.html. The syntax for encoding axioms in OPPL 2 is based on the Manchester OWL Syntax described in [10], expanded in order to accommodate variables in its constructs. Currently there are only two kinds of actions that can be performed in an OPPL 2 Script: additions and removals. A full description of OPPL2's features may be found in [4,5].

The extension to OPPL2 that makes the current work possible is the inclusion of regular expressions for variable declaration and constraint specification; a variable can now be built on the basis of a regular expression match, i.e.,: $?x : CLASS = Match("(\w+)\s\w+")$ represents the set of classes whose label is composed of two alphanumeric sequences divided by a spacing character. The same construct can be used to model a constraint, e.g.,: $SELECT\ ?x\ WHERE\ ?x\ Match("(\w+)\s\w+")$ will restrict the possible matches for $?x$ to those whose label matches the regular expression.

Java regular expressions are used to evaluate the input expression; therefore the syntax constraints and expressive power are those of Java regular expressions. Matching groups can be used, for example to build other generated variables:

[1] asserted or inferred.

$?y = create(?x.GROUPS(1))$ will use the string captured by the first group in the expression used to define $?x$.

We have already observed that a large portion of the 'binding' sub-hierarchy of GO MF conforms to the label pattern X *binding*, where X is a chemical. We can therefore create a regular expression $(\backslash w+)\backslash sbinding$ that will match said labels; the captured group will be the label of the chemical X. This value can then be used to allow for the dissection of GO MF's labels to expose the implicit semantics of the class explicitly as axioms.

3 Results

We show detailed results for only the binding part of GO MF. The full set of patterns and OPPL2 scripts; together with the ontologies may be found at http://miuras.inf.um.es/ mfoppl/.

3.1 Analysing Labels and Text Definitions for Binding Functions

Binding activities are defined in GO as "The selective, non-covalent, often stoichiometric, interaction of a molecule with one or more specific sites on another molecule". The bindings are then produced with substances or cellular components. In some cases, the binding is produced with a particular part of the substance or the component. There are different forms of binding according to the labels: binding, pairing and self-binding. The subtypes of binding activities are mainly due to the different types of substances and components that are bound.

This version of the molecular function ontology contains 1567 descendant classes of binding. 36 binding subclasses define their own subtaxonomy, and 18 do not. The *binding* subclasses have the following pattern for the label: "X *binding*", where X is the substance. In some cases the lexical pattern is "X Y *binding*", where X is the substance and Y is the type of substance; for instance for *receptor bindings* and *protein domain specific bindings*, which follow the pattern "*X domain binding*" and "*X receptor binding*", respectively, and this affects 354 (out of 398) classes in the case of receptors and 35 subclasses in the case of protein domain specific labels.

The *molecular adaptor activity* class does not follow the basic pattern above. This function is defined in GO as "the binding activity of a molecule that brings together two or more molecules, permitting those molecules to function in a coordinated way". This class has 72 descendants that follow the pattern "*X Y adaptor activity*", where X and Y are the substances that are brought together.

Another type of binding is *base pairing* that is defined in GO as "interacting selectively and non-covalently with nucleic acid via hydrogen bonds between the bases of a gene product molecule and the bases of a target nucleic acid molecule". The labels of these classes follow the pattern "base pairing with X", where X is a nucleic acid. This pattern is followed by 7 out of 9 classes in the first two

taxonomic levels. At this point, the pattern changes: "X modification guide activity", where X is a nucleic acid. The other is "X codon-amino acid adaptor activity", where X is an mRNA triplet. These classes follow the previously described molecular adaptor activity pattern.

3.2 The auxiliary ontologies

The analysis above reveals the types of biological entities implied within the labels and text definitions of GO MF. In general these are chemicals, both large and small, to which the functions apply. The following ontologies were used to provide these entities ontologically:

- Chemical Entities of Biological Interest (CHEBI)[11]: CHEBI is a freely available dictionary of molecular entities. It is an ontological classification, whereby the relationships between molecular entities or classes of entities and their parents and/or children are specified. in this work, we have used the OWL version of its release 59.
- Relation Ontology (RO)[9]: The ontology of biomedical relations provides a common set of biomedical relationships being used in the development of OBO ontologies to facilitate knowledge sharing and interoperability.
- Biochemical ontologies (http://dumontierlab.com/): These are a set of ontologies developed by Dumontier's lab whose goal is the representation of biological and scientific concepts and relations.
- Amino acid ontology(http://www.co-ode.org/ontologies/amino-acid/): This is a small ontology focused on providing the specific, explicit semantics of amino acids and their properties.
- Protein ontology: Some ontologies about proteins have been developed, although they did not provide the kind of knowledge we needed for this work. Some examples are PRO (http://pir.georgetown.edu/pro/), and the ProteinOntology (http://proteinontology.org.au/). Therefore, we developed a small protein ontology following the classification of proteins suggested in http://proteincrystallography.org/protein/.
- Enzyme ontology (http://ontology.dumontierlab.com/ec-primitive): This ontology implements the Enzyme Commission (EC) classification, which is a numerical classification scheme for enzymes, based on the chemical reactions they catalyze.
- FMA (http://www.berkeleybop.org/ontologies/owl/FMA): The Foundational Model of Anatomy ontology represents the classes and relationships necessary for the symbolic structural representation of the human body.
- Other GO ontologies: The ontologies developed by the Gene Ontology Consortium for representing cellular components and biological processes [2].

In addition to this, we also added to the ontology biochemical substances and types of substances that did not appear in any of these ontologies. In some cases, the same biochemical entity was found in more than one ontology, so different concepts had to be merged.

3.3 Binding Function Ontology Design Patterns

Binding The general axiomatisation for the binding function is:

```
binding = molecular_function and enables some
(binds some chemical_substance or binds some cellular_component)
```

The actual OPPL2 scripts for applying this axiomatisation are:

```
?x:CLASS, ?y:CLASS
BEGIN
ADD ?y subClassOf molecular_function,
ADD
?y subClassOf enables some (binds some ?x)
END;
?y is a molecular function that binds the chemical substance
or cellular component ?x
```

There, we can see how the label is processed and the linguistic expression contained in the label and that is different from "binding" is assigned to the variable ?y. This corresponds to the particular substance or cellular component that participates in the function.

```
?y:CLASS=Match("((\w+))_binding"), ?x:CLASS=create(?y.GROUPS(1))
SELECT ?y subClassOf Thing WHERE ?y Match("((\w+))_binding")
BEGIN
ADD ?y subClassOf molecular_function,
ADD ?y subClassOf enables some (binds some ?x)
END;
```


The patterns for domain and receptor bindings only change in the regular expression that is used for processing the labels to "(((\w+))_domain_binding)" and "(((\w+))_receptor_binding)" respectively.

Molecular adaptor activity The general axiomatisation for the molecular adaptor binding function is:

```
molecular_adaptor_activity= molecular_function
and enables some (adapts some molecule and
                              adapts min 2 molecule)
```

The ODP for applying this pattern of axioms is:

```
?x:CLASS, ?y:CLASS
BEGIN
ADD ?y subClassOf molecular_function,
ADD ?y subClassOf enables some (adapts some ?x and
                                  adapts min 2 molecule)
END;
?y is a molecular function that adapts ?x
```

The actual OPPL2 script that does the matching and application of axioms is:

```
?y:CLASS=Match("((\w+))_adaptor_activity"),
?x:CLASS=create(?y.GROUPS(1))
SELECT ?y subClassOf Thing WHERE ?y Match("((\w+))_adaptor_activity")
BEGIN
ADD ?y subClassOf molecular_function,
ADD ?y subClassOf enables some
                        (adapts some ?x and adapts min 2 molecule)
END;
```

Triplet codon amino acid adaptor activity This pattern allows a more precise definition that the first one. These functions are types of molecular adaptor activities. The general axiomatisation is:

```
triplet_codon_amino_acid_adaptor_activity=
molecular_function and enables some (adapts some triplet and adapts
some amino_acid )
```

The OPPL2 pattern is:

```
?x:CLASS, ?y:CLASS
BEGIN
ADD ?y subClassOf molecular_function,
ADD ?y subClassOf enables some
               (adapts some (amino_acid and recognizes some ?x))
END;
```


`?y` is a molecular function that adapts `?x` and a triplet codon-amino acid and recognizes the codon ?x. Finally, the script that executes the pattern is:

```
?y:CLASS=Match("((\w+))_codon_amino_acid_adaptor_activity"),
?x:CLASS=create(?y.GROUPS(1))
SELECT ?y subClassOf Thing
WHERE ?y Match("((\w+))_codon_amino_acid_adaptor_activity")
BEGIN
ADD ?y subClassOf molecular_function,
ADD ?y subClassOf enables some
            (adapts some (amino_acid and recognizes some ?x))
END;
```

Base pairing This biological function has been explicitly defined using the OWL axioms:

```
 base_pairing = molecular_function and
enables some (pairs some nucleic_acid and through some hydrogen_bond)
```

This general axiomatisation corresponds to the OPPL2 pattern:

```
?x:CLASS, ?y:CLASS
BEGIN
ADD ?y subClassOf molecular_function,
ADD ?y subClassOf enables some
            (pairs some ?x and through some hydrogen_bond)
END;
?y is a molecular function that pairs ?x molecules through hydrogen
bonds.
```

This pattern has been extracted from the textual definition of the function and encoded in the pattern. The pattern is executed through the script:

```
?y:CLASS=Match("base\_pairing\_with\_((\w+))"),
?x:CLASS=create(?y.GROUPS(1))
SELECT ?y subClassOf Thing
WHERE ?y Match("base_pairing\_with_((\w+))")
BEGIN
ADD ?y subClassOf molecular_function,
ADD ?y subClassOf enables some
                    (pairs some x and through some hydrogen_bond)
END;
```

Modification guide activity The general axiomatisation is:

```
modification_guide_activity
= molecular_function and enables some (pairs some nucleic_acid and
through some hydrogen_bond and guides some nucleic_acid)
```


The same axiomatisation realised as an OPPL2 pattern is:

```
?x:CLASS, ?y:CLASS, ?z:OBJECTPROPERTY
BEGIN
ADD ?y subClassOf molecular_function,
ADD ?y subClassOf enables some
(pairs some ?x and through some hydrogen_bond and ?z some ?x)
END;
?y is a molecular function that pairs ?x molecules through hydrogen
bonds and guides the ?x ?z operation.
```

The full OPPL2 script is:

```
?y:CLASS=Match("((\w+))_((\w+))_guide\_activity"),
?x:CLASS=create(?y.GROUPS(1)),
?z:OBJECTPROPERTY=create(?y.GROUPS(3))
SELECT ?y subClassOf Thing
WHERE ?y Match("((\w+))_((\w+))_guide_activity")
BEGIN
```

```
ADD ?y subClassOf molecular_function,
ADD ?y subClassOf enables some (pairs some ?x
         and through some hydrogen_bond and ?z some ?x)
END;
```

3.4 Execution of the OPPL Patterns Against Binding Functions

The version of GO used in this work has 1 567 descendant classes of the class *binding*, among which 54 are direct subclasses. The results of applying the basic pattern are shown in Table 1, where the results are grouped by binding subclass. The global result shows that 1 228 (around 78%) of the classes are matched by the pattern. This result includes the domain and receptor binding patterns. Most of the labels of the non-matched classes are also encoding other biological functions than binding, such as molecular adaptor; this is due to the multiple inheritance in GO MF. In addition, around 200 non-matched classes are types of receptor activities that have multiple parents (more than one function), so the binding pattern needs to be combined with another to accommodate this other function.

Base pairing There are 84 subclasses of *base pairing* although the pattern only matches 6, because the rest are also kinds of *molecular adaptor activity* and *guide activity*.

Molecular adaptor activity: There are 72 descendant classes and the pattern matches correctly 71. The non-matching is due to inconsistency of naming in the labels, because its label is "*X adaptor protein activity*".

Triplet codon amino acid: There are 64 descendant classes and the pattern correctly matches all of them.

Modification guide activities: There are 12 descendant classes that are correctly matched by the pattern.

In summary, these patterns match 1 336 binding activities, that is, around 85%, and up to 94% if the 157 receptor activities are not included.

3.5 Findings

The original molecular function ontology had 8 548 classes, 5 object properties, 5 data properties and 9954 subclass axioms, having ALER(+D) DL expressivity. The object properties are *part_of*, *regulates*, *negatively_regulates*, *positively_regulates* and *ObsoleteProperties* (which is used for archiving purposes). However, part_of is used in only 8 axioms of the ontology. This ontology is classified in less than 1 second (average time over 5 runs under the same conditions: 842,8 ms).

The transformed ontology has 58 624 classes, 254 object properties, 16 data properties, 107 631 subclass axioms, 264 equivalent class axioms and 488 disjoint

subclass	number of subclasses	number of correct matches
alcohol	4	4
amide	2	2
amine	29	19
antigen	5	5
bacterial	6	6
base pairing	84	0
carbohydrate	31	28
carbon monoxide	2	1
carboxylix acid	8	8
cell surface	5	5
chromatin	6	6
cofactor	19	17
DNA	78	69
drug	10	10
extracellular matrix	6	6
hormone	23	18
host cell surface	1	1
ion binding	33	30
isoprenoid	9	9
lipid	50	48
metal cluster	5	5
molecular adaptor	72	0
neurotransmitter	39	3
nucleobase	8	8
nucleoside	9	9
nucleotide	38	38
odorant	2	2
oxygen	2	1
pattern	20	14
peptide	87	28
phthalat	3	3
pigment	2	2
protein	920	766
ribonucleoprotein	5	5
RNA	135	52
tetrapyrrole	5	5
translation factor	8	2
vitamin	19	19
TOTAL	1790	1254

Table 1. Results of the application of the binding pattern by subclass.

class axioms, with SRIQ(D) DL expressivity. This ontology is classified in around 120 seconds (average time over 5 runs under the same conditions: 123 283 seconds). All the classification times have been measured using Protege 4.0.2 and the Fact++ reasoner (version 1.3.0, may 2009), and Protege has been launched with 2GB of memory.

This means that the knowledge that can be now exploited has increased by 50 076 classes, 249 object properties, 11 data properties, 97 677 subclass axioms, 264 equivalent class axioms, and 488 disjoint class axioms. Much of this knowledge is provided by the auxiliary ontologies. If we consider just the knowledge added by the execution of the patterns, the numbers are then 584 classes, 13 object properties and 3 608 subclass axioms. The object properties added correspond to the biological functions defined in the patterns such as *binds* or *formation_of*. The subclass axioms correspond to the ones added by the patterns.

Non-matched classes that have been extracted as groups in the regular expressions (mostly chemicals) are created automatically by OPPL2. These are made children of *NotInAuxiliary*. The non-matched substances include D1 Dopamine, eye lens, vasopressin, type X Y, where X is a number and Y is a substance, etc. These classes may refer to substances that exist in the auxiliary ontology but whose labels are different. This can be due to use of abbreviations (e.g., IgX vs 'immunoglobulin X', where X indicated the type of immunoglobulin, mitogen activated protein kinases vs MAPKs, etc. Otherwise meaning is embedded in the taxonomic relation. For instance there are 55 classes (grouped in *NotInAuxiliaryComplex*) whose label in the transformed ontology is 'X' and their label in the auxiliary ontologies is 'X complex'. For 14 of them, the application of a complex binding pattern would work, in the sense that we could add the suffix complex to the label. However, this would not work for 41, and it is not clear in some cases whether they refer to complexes or substances in the labels and the textual definitions.

One of the most important benefits of the transformed ontology is that we can now make more queries. The original ontology could not be asked queries such as:

1. Query 1:"Molecular functions that participate in the formation of a cellular component".
2. Query 2: "Molecular functions that bind substances that can play a chemical role"
3. Query 3:"Molecular functions that bind nitrogenous bases"

These queries can now be asked as follows:

1. Query1: molecular_function and enables some (formation_of some cellular _component). This query returns 20 direct subclasses and 30 descendants. This query benefits from using the cellular component ontology and the axiomatization provided by the patterns.
2. Query 2: molecular_function and enables some (binds some ('chemical substance' and 'has role' some 'chemical role')). This query returns 27 direct subclasses and 74 descendants. This query benefits from the axiomatization provided by the patterns and the auxiliary ontologies.

3. Query 3: molecular_function and enables some (binds some 'nitrogenous base'). This query returns 10 direct subclasses and 45 descendants. As in the previous case, this query benefits from the axiomatization provided by the patterns and the auxiliary ontologies.

3.6 Transformation times

A graphical representation of the time needed for executing the OPPL patterns is shown in Figure 1in the following order: base pairing (6), modification guide activities (12), domain binding (35), triplet codon amino acid adaptor activity(65), molecular adaptor activity (72), receptor binding (338), and binding (1228).

The increase of the time required to apply the pattern is rather stable for less than 100 classes, whereas it increases for more than 100 classes. In this particular experiment, the slope between 72 and 338 (0,139) is similar to the one between 338 and 1228 (0,144), This means that, although the figure does not allow to appreciate clearly this issue, the time increases linearly with the number of classes affected by the pattern.

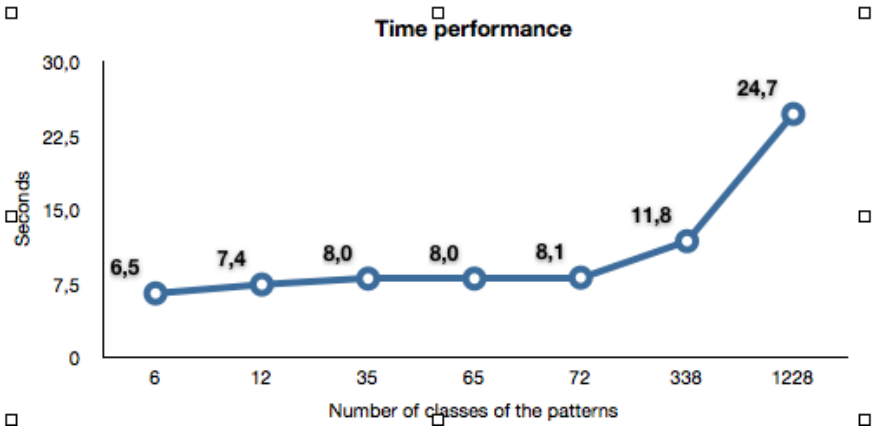

Fig. 1. Graphical representation of time performance

4 Conclusions

In this paper, the automatic transformation of a part of the molecular function ontology has been addressed by applying Ontology Design Patterns implemented using OPPL. An important goal of this work was to extract the knowledge that is embedded in the labels of the classes and this was also a limitation that we imposed ourselves in this work, since we did not attempt to capture the complete

semantics of the classes, but the one that might be extracted from the analysis of the labels.

Our OPPL scripts have been designed and executed to improve the semantic definition of the targeted molecular functions, but this does not mean that the semantics of such functions is complete. For instance, binding-related patterns provide axioms only from the binding perspective, whereas some functions may encode other biological functions. For instance, some binding functions are also related to structural molecule such as protein scaffold activities. Given the structure of the labels, only the scaffold pattern has been applied to them. This is a limitation of the label processing approach. However, since we have modelled the binding pattern, we can just apply it directly to those classes, so improving the number of binding functions modelled. This is also one of the benefits of applying OPPL patterns, since it allows for executing to all the classes that hold some criteria or to individually selected ones.There are other cases in which binding functions are also transport functions. The same process could be followed but we have not included the modelling of transport functions in this study.

This work has exploited the relatively systematic naming within life science ontologies and the naming conventions of the OBO ontologies in particular. Without these aspects this technique is much less applicable. Some of the non-matches do, however, highlight defects in some entity's labels and this is a useful side-effect. That such consistent naming and naming conventions can be thus exploited can also act as a spur to adopt such best practices.

It should be pointed out that the auxiliary ontologies are far from being perfect, hence they can be improved in different ways. On the one hand, its content can be improved by including, for instance, more substances. Nevertheless, those will be eventually added with the evolution of the source ontologies. On the other hand, the quality of myauxiliarontology can be enhanced. First, its semantics can be optimized by increasing the axiomatization concerning types of substances. In this ontology, we can find that some types of substances have been defined as equivalent classes but some have not. The consequences of this is that the results of the queries are potentially suboptimal. Second, this ontology has been obtained by the partial, manual integration of a set of ontologies. This result could be improved if ontology mapping and alignment techniques were used for supporting this integration process.

This work has shown OPPL to be useful for applying patterns to ontologies, since it provides a flexible way of adding axioms to ontologies in a systematic way. It has been capable of adding axioms to more than 1 600 classes by applying the patterns in a reasonable time. This time is mainly due to the need for matching the label patterns, which have to be compared for every class. The time performance also suggests an acceptable behaviour for larger numbers of classes matched by patterns. Nevertheless, this is one of the first applications of OPPL for executing label-based patterns and the management of regular expressions in OPPL can still improve.

In addition to this, we can see that regular expressions for dissecting labels or identifiers on classes to expose implicit axioms through the application of ODPs

can be highly effective and relatively low cost. Therefore, we plan to continue the definition of the patterns for the rest of families of functions contained in GO MF.

Acknowledgements: JTFB has been supported by the Spanish Ministry of Science and Innovation through the grants JC2008-00120 and TSI2007-66575-C02-02.

References

1. Ogren, P., Cohen, K., Acquaah-Mensah, G., Eberlein, J., Hunter, L.: The Compositional Structure of Gene Ontology Terms. In: Pacific Symposium on Biocomputing. Volume 9. (2004) 214–225
2. GOConsortium: Gene ontology: tool for the unification of biology. Nature Genetics **25**(1) (2000) 25–29
3. Presutti, V., Gangemi, A.: Content ontology design patterns as practical building blocks for web ontologies. In Li, Q., Spaccapietra, S., Yu, E.S.K., Olivé, A., eds.: Conceptual Modeling - ER 2008, 27th International Conference on Conceptual Modeling, Barcelona, Spain, October 20-24, 2008. Proceedings. Volume 5231 of Lecture Notes in Computer Science., Springer (2008) 128–141
4. Iannone, L., Rector, A.L., Stevens, R.: Embedding knowledge patterns into owl. In Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E.P.B., eds.: ESWC. Volume 5554 of Lecture Notes in Computer Science., Springer (2009) 218–232
5. Egaña, M., Rector, A.L., Stevens, R., Antezana, E.: Applying ontology design patterns in bio-ontologies. In Gangemi, A., Euzenat, J., eds.: EKAW. Volume 5268 of Lecture Notes in Computer Science., Springer (2008) 7–16
6. Solomon, W.D., Roberts, A., Rogers, J.E., Wroe, C.J., Rector, A.L.: Having our cake and eating it too: How the GALEN Intermediate Representation reconciles internal complexity with users' requirements for appropriateness and simplicity. In Overhage, J.M., ed.: Proceedings of the 2000 American Medical Informatics Association Annual Symposium (AMIA 2000), Los Angeles, American Medical Informatics Association, Hanley and Belfus Inc. (2000) 819–823
7. Wroe, C., Stevens, R., Goble, C., Ashburner, M.: A Methodology to Migrate the Gene Ontology to a Description Logic Environment Using DAML+OIL. In: 8th Pacific Symposium on biocomputing (PSB). (2003) 624–636
8. Rector, A.L.: Modularisation of domain ontologies implemented in description logics and related formalisms including owl. In: Proceedings of the 2nd International Conference on Knowledge Capture. (October, 2003, Sanibel Island, USA. 2003)
9. Smith, B., Ceusters, W., Klagges, B., Köhler, J., Kumar, A., Lomax, J., Mungall, C., Neuhaus, F., Rector, A.L., Rosse, C.: Relations in biomedical ontologies. Genome Biology **6** (2005) R46
10. Horridge, M., Drummond, N., Godwin, J., Rector, A., Stevens, R., Wang, H.: The Manchester OWL Syntax. In: Proceedigns of OWLED 2006 OWL: Experiences and Directions, Athens GA, USA. (2006)
11. de Matos, P., Alcántara, R., Dekker, A., Ennis, M., Hastings, J., Haug, K., Spiteri, I., Turner, S., Steinbeck, C.: Chemical entities of biological interest: an update. Nucleic Acids Research **38**(Supplement 1:D249) (2010)